OCR
Oxford Cambridge and RSA

An OCR endorsed teaching and learning tool

OCR A Level Computer Science

H446 – Paper 1

2

Writing and following algorithms

Unit 3 Software development

PG ONLINE

Objectives

- Understand the term “algorithm”
- Learn how to write algorithms using pseudocode
- Learn how to interpret algorithms and determine their purpose

Algorithm

- An algorithm is a set of instructions to solve a problem or complete some well-defined task in a finite number of steps
- A recipe for chocolate cake, or a knitting pattern, are algorithms of a kind

Computational algorithms

- Give some examples of computational algorithms

There are thousands of real-world problems to be solved using different algorithms; some of them still unsolved!

Problems solved by algorithms

- **Routing** problems:
  - Routing packets of data around the Internet by the shortest route
  - Finding the shortest route for a salesman to cover his territory
- **Timetabling** commercial aircraft crews so that they do not exceed their permitted flight hours
- **Searching** information on the Internet or from a database
- **Encrypting** communications so that they cannot be hacked
- **Sorting** large amounts of data
- **Writing a compiler** program to translate a high level

What are the properties of a good algorithm?

- Discuss in pairs and come up with a list

A good algorithm:

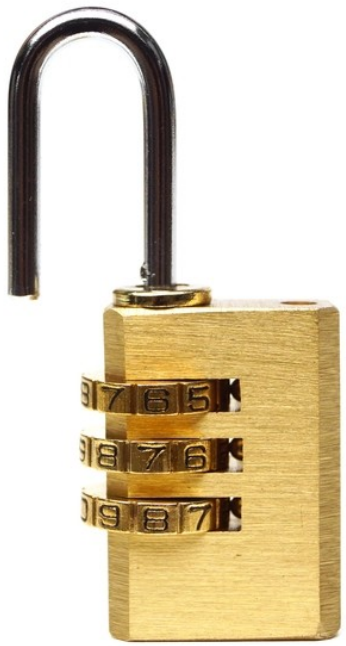
- has clear and precisely stated steps that produce the correct output for any set of valid inputs
- should allow for invalid inputs
- must always terminate at some point
- should perform the task efficiently, in as few steps as possible
- should be designed in such a way that other people will be able to understand it and modify it if necessary

Using pseudocode

- Pseudocode is a halfway house between English statements and program code
- To a large extent, it is independent of any particular programming language
  - Are there rules for writing pseudocode?
  - Is pseudocode in some ways dependent on the programming language you intend to use?

Worksheet 2

- Try **Task 1** on **Worksheet 2**



Sorting algorithms

- There are many different sorting algorithms, from very simple ones which execute very slowly, (e.g. an insertion sort) to more complex ones which execute very fast (e.g. merge sort)
- On a given computer, a fast sort algorithm may sort ten million numbers in say 20 minutes, whereas an inefficient algorithm may take more than a month to sort the same data

Bubble sort

- This is one of the simplest sorts to understand
- Starting with the first item in the list, each item is compared with the adjacent item and swapped if it is larger
- At the end of the first pass, the largest item ‘bubbles’ up to the end of the list

Pseudocode for bubble sort

```
names = [“Jane”, “Fred”, “Vicky”, “Eric”, “Bella”, “Millie”]
numItems = len(names)
for i = 0 to numItems - 2
    for j = 0 to (numItems - 2 - i)
        if names[j] > names[j + 1]
            swap the names in the array
        endif
    next j
next i
```

Bubble sort

- Suppose you have a list to be sorted alphabetically

  names = [“Henry”, “Steve”, “Julie”, “Ava”, “Tom”, “Olivia”]

- After one pass though the bubble sort algorithm, what order will the names be in?
- How many passes will be needed to sort the list into ascending sequence?
- Write an algorithm to swap two items names[j] and names[j + 1]

Searching algorithms

- Two of the most common searching algorithms are the linear search and the binary search
- The linear search starts at the beginning of the list and examines every item
- The binary search uses the “divide and conquer” strategy to halve the search area each time an item is examined
  - It can only be used if the items are in sorted order – they could be alphabetic or numeric, for example

Binary search algorithm

```
# low and high are the indices of the first and
last
# items in the list
found = False
while found == False AND high >= low
   middle = (low + high) div 2
   if list(middle) = x then
      found = True
   else
      if list(middle) > x then
         high = middle – 1
      else
         low = middle + 1
      endif
   endif
endwhile
```

Try it out!

- Ask a partner to think of a number between 1 and 100
- For each guess you make, your partner will tell you whether you are too high, too low or correct
- How many guesses will you need?
- How many if the number is between 1 and 1024?
- How many guesses, on average, using a sequential search?

Evaluating a program

- You have seen that some algorithms are much more efficient than others at solving a problem
- What about program code?
- What makes one program “better” than another?
  - Suggest ways of making your programs look like the work of
    a professional!

Writing “good” programs

- Use comments to document your programs
- Use a standard for identifiers – e.g.
  - start all variable names with a lowercase letter
  - use CamelCaps rather than spaces or underscores
- Use properly indented code
- Make sure your program does not contain statements that are not needed
- Use a modular structure, with no module longer than a page of code, performing a discrete task

Following an algorithm

- It is sometimes quite difficult to follow through an algorithm to determine its purpose or to find a
logic error
- A trace table is a useful aid
  - Draw a table with columns for each variable in the order in which they appear in the program, and a column for output
  - Follow through the algorithm line by line and fill in the value of a variable whenever it changes

Creating a trace table

- Trace through this code using the trace table:

```
a = 21
b = 15
while a != b
  if a > b then
    a = a – b
  else
    b = b – a
  endif
endwhile
print (a)
```

a	b	a != b	a > b	output
21	15	True	True	

Creating a trace table

- Trace through this code using the trace table:

```
a = 21
b = 15
while a != b
  if a > b then
    a = a – b
  else
    b = b – a
  endif
endwhile
print (a)
```

a	b	a != b	a > b	output
21	15	True	True	
6			False	

Creating a trace table

- Trace through this code using the trace table:

```
a = 21
b = 15
while a != b
  if a > b then
    a = a – b
  else
    b = b – a
  endif
endwhile
print (a)
```

a	b	a != b	a > b	output
21	15	True	True	
6			False	
	9			

Creating a trace table

- Trace through this code using the trace table:

```
a = 21
b = 15
while a != b
  if a > b then
    a = a – b
  else
    b = b – a
  endif
endwhile
print (a)
```

a	b	a != b	a > b	output
21	15	True	True	
6			False	
	9			
	3		True	

Creating a trace table

- Trace through this code using the trace table:

```
a = 21
b = 15
while a != b
  if a > b then
    a = a – b
  else
    b = b – a
  endif
endwhile
print (a)
```

a	b	a != b	a > b	output
21	15	True	True	
6			False	
	9			
	3		True	
3		False		3

  - What does the algorithm calculate?

Worksheet 2

- Try the questions in **Task 2** of the worksheet

Plenary

- Pseudocode is useful for designing algorithms
- Sorting and searching and two very common operations in data processing, and efficient algorithms can greatly reduce execution times
- Trace tables are useful for following through algorithms
- Designing and tracing through algorithms are two vital skills for computer scientists

Copyright

Licence agreement

PG ONLINE